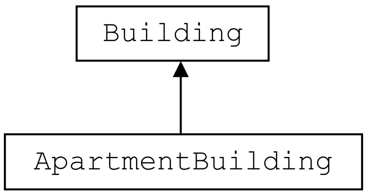
This is the Apartment Building project, which deals with inheritance and class hierarchy. To get started on the project, download, extract and save the Apartment Building project to your computer. Then work through the following instructions to complete the requirements. Note that you can test your work by running `ApartmentBuildingTest` to see how you are doing.

When you are finished, please submit a screenshot of the `ApartmentBuildingTest` success message.

Consider a hierarchy of classes used by a power company to keep track of the buildings where they supply electricity. The hierarchy is represented by the following diagram:



In other words, an `ApartmentBuilding` is a subclass of `Building`.

A building is represented by the class defined below.

```
public class Building
{
      public static final double RATE = 3.25;
      private String address;
      private double wattHours;   // units of electricity used in 1 month

      public Building(String address)
{
      this.address = address;
      wattHours = 0.0;
}

      // returns the amount owed by this building
      public double amountOwed()
      { /* implementation not shown */ }

}
```

An `ApartmentBuilding` is different from a regular building because instead of keeping track of the watt hours used for the whole building, it needs to keep track of the watt hours used by each of the individual apartments in the building.

In the Bluej project, write a complete declaration of class `ApartmentBuilding` as follows. (Be sure to compile and run the tester after each step.)

1. Create a new `ApartmentBuilding` class and then modify the class header. (Recall that an `ApartmentBuilding` is a `Building`.) Fix the problem if the tester says it is not a Building. (It is likely that your code will not compile at this point.)

2. Next, write a constructor with two parameters: the address of the apartment building (which is a `String`), and the number of apartments (which is a whole number). The constructor should initialize the building's address stored in the `Building` class.

If you are having difficulties compiling your code at this point, bear in mind that the parent class needs to be instantiated before each child (just like a family). Have a look at the parent class (`Building`). Notice the `Building` constructor that requires a `String address` as a parameter. That means every `Building` object requires an address when it is instantiated.

Therefore, because `ApartmentBuilding` is a `Building`, the `ApartmentBuilding` constructor must first pass an address to its parent class. This must be done in the first line of the constructor. Can you think of a `super()` way to do this?

3. Add an instance variable (an array) to be used to store the apartments' watt hours. What data type does it need to be to correspond with the `Building` instance variable `wattHours`?

4. Now back to the constructor, which should initialize the array to be big enough to store watt hours for each apartment in the building.

5. Add an overridden implementation of `amountOwed` that returns the amount of money owed by the entire apartment building for the electricity used (calculated by multiplying the sum of the `wattHours` for each apartment in the entire apartment building by the `RATE`).

Make sure the tester reports that the `ApartmentBuilding` class is correct before moving on.

Finally, consider the following declaration for the `ServiceArea` class. A `ServiceArea` represents an entire area being served by this power company.

```
public class ServiceArea
{
      private ArrayList<Building> allBuildings;   // a list of Buildings

      public ServiceArea()
      {
            allBuildings = new ArrayList();
      }

      public double totalSales()
      {
            /* to be completed */
      }
}
```

6. Write the `totalSales` method of class `ServiceArea`. The method should return the total amount of money owed by all of the buildings in the `allBuildings` ArrayList. You will need to iterate through `allBuildings`, getting each `Building` object from the list, and then asking each `Building` its `amountOwed`. Return the total of `amountOwed`.